UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/786,843	02/25/2004	Jose German Rivera	200312292-1	2936

22879 7590 04/17/2009

HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400

EXAMINER	
WEI, ZHENG	
ART UNIT	**PAPER NUMBER**
2192	

NOTIFICATION DATE	DELIVERY MODE
04/17/2009	ELECTRONIC

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Notice of the Office communication was sent electronically on above-indicated "Notification Date" to the following e-mail address(es):

JERRY.SHORMA@HP.COM
ipa.mail@hp.com
jessica.l.fusek@hp.com

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 10/786,843
Filing Date: February 25, 2004
Appellant(s): RIVERA ET AL.

Randy A. Noranbrock
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed 1/15/2009 and 11/25/2008 appealing from the Office action mailed 07/25/2008.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief. The real party in interest is Hewlett-Packard Development Company, L.P., a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249 Houston, TX 77070, U.S.A. (hereinafter "HPDC"). HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, LLC.

(2) Related Appeals and Interferences

The examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

Mickey Williams, "Microsoft Visual C# .NET", published on May 15, 2002 by Microsoft Press, Chapter 9, pages 1-27

GNU "The GNU C Library" section "Explicitly Checking Internal Consistency", Published online 08/12/1999 pages 1-2 [Retrieved from: http://mada.kth.se/kurser/master/intro/libc/libc_29.html]

PHP Manual, Edited by Bakken et al., on 05/30/2003, pages 1-2, [Retrieved online: http://web.archive.org/web/20030608063729/us2.php.net/manual/en/]

7,146,473	Cantrill	12-2006

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

A. Claims 1, 4, 5, 7-11, 14, 15, 17-21, 24, 25, 27-31, 34, 35, 37-41 and 44-49 are rejected under 35 U.S.C. 103(a) as being unpatentable over by Williams (Mickey Williams, Microsoft® Visual C#™ .NET) in view of GNU (The GNU C Library, Section “Explicitly Checking Internal Consistency”) and in further view of PHP (PHP Manual, Section “assert_options”

Claim 1:

Williams discloses a method for monitoring (debugging and tracing) computer software comprising:

- receiving an assertion from an executing process (see for example, p.11, line 3, "When a DefaultTraceListerner object detects that the Assert method has been called from a server process"), but does not explicitly discloses wherein the executing process is integral to an operating system. However, GNU in the same analogous art of application/operating system error checking discloses using assert method in the operating system and report execution error (see for example, p.1, section "Explicitly Checking Internal Consistency", first paragraph and p.2 third paragraph, "check for an error return from an operating system function"). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to put the assertion in the operation system code to monitor the error from the system call. One would have been motivated to do so to display all the error information and further help to debug problem as suggested by GNU (see for example, p.1, section "Explicitly Checking Internal Consistency", first paragraph, "These kinds of checks are helpful in debugging problem…"). wherein receiving an assertion comprises:
  - receiving an assertion request (see for example, p.11, line 3, "When a DefaultTraceListerner object detects that the Assert method has been called from a server process");

    But neither of them explicitly discloses about recognizing a type for assertion request. However, PHP in the same analogous art of assertion, discloses a assert control option (see for example, p.1, Table 1. Assert

Options, “assert.active” and “default value 1” for enabling assert() evaluation and related text). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to assertion control option with assert method in Williams to control monitoring process. One would have been motivated to do so to set the various assert() control options as suggested by PHP (see for example, p.1, “Using assert_options() you may set the various assert() control options…”) or

- accepting the assertion request when the determined component has assertion requests enabled (see rejection above for the assert_option is set to 1 (enable assert() evaluation)).

- recording the assertion when the assertion is violated (see for example, p.10-11, figure 9-3 ”Dialog box generated by trace and debug output with the Assert method” and related text, also see p.10, section “Asserting That Expressions Are True”, lines 15-16, The Assert method is used to display an error message when a condition that’s expected to evaluates as true evaluates as false.”); and
- allowing the executing process to continue execution (see for example, p.10-11, figure 9-3”Dialog box generated by trace and debug output with the Assert method” and button labeled as “Ignore”).

Claim 4:

Williams further discloses the method of claim 1 wherein recording the assertion comprises recording a datum that includes at least one of: type of assertion, sequence number of the assertion, time at which the assertion occurred, identification of processor that produced the assertion, identification of process that produced the assertion, identification of the thread that produced the assertion, text of the assertion, stack trace, source line containing the assertion, and file name of the source containing the code that generated the assertion (see for example, p.10-11, figure 9-3 "Dialog box generated by trace and debug output with the Assert method" and related text, also see p.10, lines 1-2, "As you can see, this dialog box includes call stack information when available. Where debug symbols are available, the stack trace includes file name and line number information.")

Claim 5:

Williams also discloses the method of claim 1 wherein recording the assertion comprises writing information regarding the assertion violation to a computer readable medium (see for example, p.9, lines 13-15, "The .NET Framework includes classes to control trace and debug output message and to write output message to files, streams, and event log.").

Claim 7:

Williams further discloses the method of claim 1 further comprising:

- accepting a command from at least one of a control console and a network connection (see for example.p.203, Figure 9-1. "The build property page for a project, on which new symbols are defined", "TraceDemo Property Pages", "Configuration Properties", also see p.13, example configuration file: *SwitchText.exe.config*); and
- updating an enable condition for an assertion class according to the command (see for example, p.13, line 41-p.14, line 2, "Switches are controlled by adding XML element nodes inside the switches element, Multiple switch objects can be configured through a configuration file by adding additional elements to the switches node." , "BooleanSwitch objects are disabled by default and are enabled if they're assigned a nonzero value in a configuration file.")

Claim 8:

Williams further discloses the method of claim 1 further comprising generating an error report according to the recorded assertion (see for example, p.11, lines 7-17. "The Assert method has three versions", "The most basic version simply accepts an expression that triggers an assertion failure message", "The second version of Assert accepts a second parameter that serves as a short error message describing the assertion violation", "The third version of Assert accepts a third parameter that includes detailed information about the assertion violation")

Claim 9:

Williams also disclose the method of claim 8 further comprising dispatching the error report to a real-time assertion monitor (Visual Studio output window) (see for example, p.11, lines 5-6, "Instead, it writes the output message to the Visual Studio Output window and any other debuggers currently accepting output from the Microsoft Win32 OutputDebugString function.").

Claim 10:

Williams further discloses the method of claim 8 wherein generating an error report comprises: retrieving an assertion violation parameter including at least one of: type of assertion, sequence number of the assertion, time at which the assertion occurred, identification of processor that produced the assertion, identification of process that produced the assertion, identification of the thread that produced the assertion, text of the assertion, stack trace, source line containing the assertion, and file name of the source containing the code that generated the assertion; and generating a report file comprising page description statements according to the assertion parameter (see for example, p.10-11, figure 9-3 "Dialog box generated by trace and debug output with the Assert method" and related text, also see p.10, lines 1-2, "As you can see, this dialog box includes call stack information when available. Where debug symbols are available, the stack trace includes file name and line number information.")

Claims 11, 14, 15, 17-20 and 45:

Claims 11-15, 17-20 and 45 are apparatus version of the claimed method addressed in claims 1-5, 7-10 and 44 above for monitoring computer software, wherein such an apparatus/computer system is deemed to be inherent to produce, such as Figure 9-3 dialog box and word above. Therefore, these claims are also anticipated by Williams.

Claims 41 and 49:

See the rejection in claim 1 above.

Claim 44:

Williams also discloses the method of Claim 1 wherein the assertion request type is one of a group of defined assertion macro names (property) (see for example, p.13, section "Using the BooleanSwitch Class", lines 20-21, "The BooleanSwitch class is used to created simple Switch objects that can be either enable or disabled", also see p.22-24, example code)

Claim 48:

Williams also discloses the apparatus for monitoring computer software of Claim 41 wherein the assertion request type is one of a group of defined assertion macro names (property) (see for example, p.13, section "Using the BooleanSwitch Class", lines 20-21, "The BooleanSwitch class is used to created

simple Switch objects that can be either enable or disabled", also see p.22-24, example code).

Claims 21, 24, 25, 27-30 and 46:

Claims 21-25, 27-30 and 46 claim a computer software monitoring system comprising: memory capable of storing instructions; processor capable of executing instructions stored in the memory; and software monitor instruction sequence that, when executed by the processor, minimally causes the processor to: receive an assertion from an executing process, record the assertion, and allow the executing process to continue execution. This is a product version of method claims discussed in claims 1-5 and 7-10 above respectively. It is well known in the computer art that the method can be practiced by the computer system to perform the same functionality. Therefore, these claims are also unpatentable over Williams, GNU and PHP.

Claims 31, 34, 35, 37-40 and 47:

Claims 31-35, 37-40 and 47claim a computer-readable medium having computer-executable instructions for performing a method for monitoring computer software. This is another product version of method claims discussed in claims 1-5 and 7-10 above respectively. It is well known in the computer art that the method can be stored and practiced in the computer-readable medium. Therefore, these claims are also unpatentable over Williams, GNU and PHP.

B. Claims 6, 16, 26 and 36 are rejected under 35 U.S.C. 103(a) as being unpatentable over Williams (Mickey Williams, Microsoft® Visual C#™ .NET) in view of GNU (The GNU C Library, Section "Explicitly Checking Internal Consistency") in further view of PHP (PHP Manual, Section "assert_options") and in further view of Cantrill (Bryan M, Cantrill, US 7,146,473)

Claim 6:

Williams, GNU and PHP disclose the method of claim 1 wherein recording the assertion comprises writing information regarding the assertion violation to output device, but does not explicitly disclose the output is a circular buffer. However, Cantrill in the same analogous art of a mechanism for ring buffering (circular buffer) in an arbitrary-action tracing framework (see for example, col.7, lines 15-17, "Embodiments of the invention provide a means for implementing a ring buffer scheme in arbitrary-action tracing frameworks which have variable length records."). Therefore, it would have been obvious to one having ordinary skill in the art at the time the invention was made to use circular buffer to store the output message. One would have been motivated to do so to keep the most recent recorded message in the fix sized buffer as suggested by Cantrill (col.1, lines 28-30, "one may which only want to keep the most recent data. To allow for this, tracing frameworks have historically implemented ring buffer.")

Claims 16, 26 and 36:

Claims 16, 26 and 36 are different product versions of method claim 6. It is well known in the computer that these products can be used to practice or perform

the method discussed in claim 6 above. Therefore these claims are also unpatentable over Williams, GNU and PHP in view the teachings of Cantrill.

(10) Response to Argument

1. Neither Williams nor GNU disclose or suggest "receiving an assertion from an executing process, wherein the executing process is integral to an operating system". Because, GNU directed to user application and not to an operating system process – emphasis added.

In response, first of all, Williams discloses receiving an assertion from an executing process (server process) - See, for example, p.11, 2nd paragraph, "When a DefaultTraceListerner object detects that the *Assert* method has been called from a server process…" – emphasis added, and that is to say, in order for the server process to call for/invoke the *Assert* method, the executing process (server process) has to be run under (integral to) an operating system so that the executing server process can call for/invoke the *Assert* method at executing time. Accordingly, when the executing process, under the running operating system as a whole, would receive and execute the called *Assert* method, and if there are any violated assertions, the error message can be displayed/printed/generated by using operating system's function call - See, for example, p.11, 2nd paragraph, "… Instead, it writes the output message to the Visual Studio Output window and any other debuggers currently accepting output from the Microsoft Win32 *OutputDebugString* function." Secondly, GNU also discloses an assertion

method (assert_perror() macro) which is used to check an error return from an operating system function (integral to an operating system) – emphasis added - See, for example, p.2, second paragraph, states

> "Sometimes the 'impossible' condition you want to check for is an error return from an operating system function. Then it is useful to display not only where the program crashes, but also what error was returned. The assert_perror macro makes easy."

It also should be noted that in order to check the error return from the operating system, the executing process which issues/invokes/calls for the assert_perror() macro has to interact with/run under (integral to) an operating system as one of the executing processes controlled by the operating system. Therefore, the executing process of Williams and/or GNU when used with assertion methods such as assert_perror() or *Assert* method, indeed, teach the recited limitation about the executing processing has to be integral to an operating system.

2. GNU fails to disclose receiving an assertion from an executing process integral to an operating system. Because there is no disclosure of an assertion from the operating system and GNU states that on an assertion being given invalid input, a program will abort – emphasis added.

In response, as discussed above, Williams discloses receiving an assertion from an executing process (server process) - See, for example, p.11, 2nd paragraph, "When a DefaultTraceListerner object detects that the *Assert* method has been

called from a server process" – emphasis added, and that is to say in order for the server process to call for/invoke the *Assert* method, the executing process (server process) has to be run under (integral to) an operating system so that the executing server process can call for/invoke the *Assert* method at executing time. Again, accordingly, the operating system and executing process as a whole, would receive/execute the called *Assert* method. And, as a result of executing the *Assert* method, if there are any violated assertions, the error message can be displayed/printed by using the operating system - See, for example, p.11, 2nd paragraph, "the Microsoft Win32 *OutputDebugString* function". It should be noted that the plain language of the claim merely called for "the executing process is integral to an operating system" and that does not limit the claimed "assertion" have to be from the operating system software itself. It can be reasonable interpreted as an assertion from a process of a program running or executing under (integral to) the operating system which includes any assertions from any executing processes of an application program. Therefore, GNU's assertion application process with assertion method such as assert_perror() when executed by/run under (integral to) the operating system, has to be received and processed by the operating system and executing process as a whole, and that where the executing process is running under control of (integral to) the operating system. Moreover, GNU further discloses an option to continue running the program after printing its error messages - See, for example, p.2, second paragraph from the bottom, states

> "What's more, your program should not abort when given invalid input, as assert would do – it should exit with nonzero status ... after printing its error messages, or perhaps read another command or move on to the next input file."

Furthermore, Williams also discloses the same limitation about continue execution the executing process after printing out error message –See for example, p. 11, Figure 9-3, "Dialog box generated by trace and debug output with the Assert method" shows a dialog box with assertion failed error message and further provides three user selectable option buttons: "Abort", "Retry" and "Ignore" which mean "Assertion Failed: Abort=Quit, Retry=Debug, Ignore=Continue" and that is to say, the user can select or click the "Ignore" button to continue executing the process after the printed out error message as showed in the dialog box. Therefore, both GNU and Williams disclose receiving an assertion from an executing process integral to an operating system and also allowing the executing process to continue execution.

3. GNU explicitly teaches a program aborting without proceeding to continue execution as a result of an assert occurrence – emphasis added.

In response, it should be noted that the Appellants merely rely on GNU's disclosure at page 1, wherein execution of the program is aborted on an

evaluation of an expression to false (zero). However, GNU at page 2 further teaches another option to continue running the program after printing its error messages - See, for example, p.2, second paragraph from the bottom,

> "What's more, your program should not abort when given invalid input, as assert would do – it should exit with nonzero status…after printing its error messages, or perhaps read another command or move on to the next input file."

Furthermore, Williams also discloses the same limitation about continue execution the executing process after printing out error message –See for example, p. 11, Figure 9-3, "Dialog box generated by trace and debug output with the Assert method" shows a dialog box with assertion failed error message and further provides three user selectable option buttons: "Abort", "Retry" and "Ignore" which mean "Assertion Failed: Abort=Quit, Retry=Debug, Ignore=Continue" and that is to say, the user can select or click the "Ignore" button to continue executing the process after the printed out error message as showed in the dialog box. Therefore, either GNU or Williams discloses the options to allow a program aborting or continuing execution as a result of an assert occurrence.

4. The PTO has failed to set forth a prima face case of obviousness

In response, as discussed above, Williams and GNU disclose receiving an assertion from an executing process integral to an operating system and also allowing the executing process to continue execution as indicated by GNU and GNU further points out that "Sometimes the 'impossible' condition you want to check for is an error return form an operating system function. Then it is useful to display not only where the program crashed, but also what error was returned" – See, for example, p.2, second paragraph. Therefore, previous office action has set the prima facie case of obviousness with respect to the combination of Williams and GNU.

5. PHP fails to disclose recognizing an assertion request type – emphasis added.

In response, Appellants submit that PHP merely describe the setting/getting of various assert flags related to control options of an assert. However, PHP or even in light of the Appellants' own specification (there are only two assertion types, i.e., enabled yes/no - See Fig.2, step 45, TYPE ENABLED YES/NO) discloses including assertion type enabled or disabled such as "**assert_options ()** will return the original setting of any option or FALSE on errors" (See, for example, PHP, p.1, last line); also see Table 1, Assert Options, with ini-parameter "assert.active" and default setting, e.g., 1, and that is to say the assertion is controlled by the active types including enable and disable, as option type now is setting as ASSERT_ACTIVE, assert.active=1, which enable **assert()**

evaluation. Therefore, use the assertion request type to control enabled/disabled assertion during execution is taught by PHP.

6. PHP fails to disclose a determination of the component that sourced the assertion request – emphasis added.

In response, it should be noted that the plain language of the claim merely called for only one requirement, for example, "performing at least one of" – emphasis added. Therefore, as PHP performed "recognizing an assertion request type corresponding to the assertion request", it discloses one of said claim limitations as recited.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

/Zheng Wei/

Examiner, Art Unit 2192

Conferees:

/Tuan Q. Dam/
Tuan Q. Dam
Supervisory Patent Examiner, Art Unit 2192

/Lewis A. Bullock, Jr./
Lewis A. Bullock, Jr.
Supervisory Patent Examiner, Art Unit 2193